

# A New Bound on the Performance of the Bandwidth Puzzle

Zhenghao Zhang  
Computer Science Department  
Florida State University, Tallahassee, FL, USA  
zzhang@cs.fsu.edu

**Abstract**—A bandwidth puzzle was recently proposed to defend against colluding adversaries in peer-to-peer networks. The colluding adversaries do not do actual work but claim to have uploaded contents for each other to gain free credits from the system. The bandwidth puzzle guarantees that if the adversaries can solve the puzzle, they must have spent substantial bandwidth, the size of which is comparable to the size of the contents they claim to have uploaded for each other. Therefore, the puzzle discourages the collusion. In this paper, we study the performance of the bandwidth puzzle and give a lower bound on the *average* number of bits the adversaries must receive to be able to solve the puzzles with a certain probability. We show that our bound is tight in the sense that there exists a strategy to approach this lower bound asymptotically within a small factor. The new bound gives better security guarantees than the existing bound, and can be used to guide better choices of puzzle parameters to improve the system performance.

## I. INTRODUCTION

A key problem in peer-to-peer (p2p) based content sharing is the incentive for peers to contribute bandwidth to serve other peers [16]. Without a robust incentive mechanism, peers may choose not to upload contents for other peers, causing the entire system to fail. In many applications, a peer's contribution is measured by the number of bits it uploaded for other peers. It is difficult to measure the contribution because peers may collude with each other to get free credits. For example, if Alice and Bob are friends, Alice, without actually uploading, may claim that she has uploaded a certain amount of bits for Bob. Bob, when asked about this claim, will attest that it is true because he is Alice's friend. Therefore, Alice gets free credits.

With the current Internet infrastructure, such collusions are difficult to detect, because the routers do not keep records of the traffic. Recently, a bandwidth puzzle scheme has been proposed solve this problem [14]. In the bandwidth puzzle scheme, a central credit manager, called the *verifier*, is assumed to exist in the network. The verifier issues puzzles to suspected nodes, called *provers*, to verify whether the claimed transactions are true. To be more specific, when the verifier suspects a set of provers for certain transactions, it issues puzzles *simultaneously* to all the involved provers, and asks them to send back answers within a time threshold. The puzzle's main features are (1) it takes time to solve a puzzle and (2) a puzzle can be solved only if the prover has access to the contents. To

illustrate the basic idea of the puzzle, consider the previous simple example with Alice and Bob. The verifier issues two puzzles, one to Alice and one to Bob. As Alice did not upload the content for Bob, Alice has the content but Bob does not. When received the puzzles, Alice can solve hers and send the answer to the verifier before the threshold but not Bob. Bob also cannot ask help from Alice, because Alice cannot solve two puzzles within the threshold. Given this, Bob will fail to reply with the answer of the puzzle and the verifier will know that the transaction did not take place.

The bandwidth puzzle is most suited for live video broadcast applications, where fresh contents are generated constantly [14]. The verifier can naturally reside in the source node of the video, and the puzzle is based on the unique content currently being broadcast, such that there can be no existing contents downloaded earlier to solve the puzzles. The construction of bandwidth puzzle is simple and based only on hash functions and pseudorandom functions. In [14], the puzzle scheme was implemented and incorporated into a p2p video distributing system, and was shown to be able to limit collusions significantly. An upper bound was also given for the expected number of puzzles that can be solved given the limit of the number of bits received among the adversaries. However, the bound is "loose in several respects," as stated by the authors, because its dominating term is quadratic to the number of adversaries such that it deteriorates quickly as the number of adversaries increases. In this paper, we give a much improved bound on the performance of the puzzle. The new bound gives the *average* number of bits the adversaries must have received if they can solve the puzzles with a certain probability. As we will prove, the average number of bits the adversaries receive is linear to the number of adversaries for all values of adversaries. It is also asymptotically tight, in the sense that there exists a strategy that achieves this bound asymptotically within a small factor. The improved bound leads to more relaxed constraints on the choice of puzzle parameters, which should in turn improve the system performance.

The rest of this paper is organized as follows. Section II describes the construction of the puzzle. Section III gives the proof of the new bound. Section IV discusses the practical puzzle parameters and shows how a simple strategy approaches the bound. Section V discusses related works. Section VI concludes the paper.

$n$	The number of bits in the content
$k$	The number of indices in an index set
$L$	The number of index sets in a puzzle
$z$	The number of puzzles sent to a prover
$\theta$	The time threshold to solve the puzzles

TABLE 1  
LIST OF PUZZLE PARAMETERS

## II. THE CONSTRUCTION

In this section, we describe the construction of the puzzle. The puzzle construction is largely the same as [14] except one difference: allowing repeated indices in one index set (the definition of index set will be given shortly), which simplifies the puzzle construction. We first give a high-level overview of the puzzle construction as well as introducing some notations. The main parameters of the puzzle are listed in Table I.

### A. A High-level Description

The content being challenged is referred to simply as *content*. There are  $n$  bits in the content, each given a unique index. An *index set* is defined as  $k$  ordered indices chosen from the  $n$  indices. Each index set defines a string denoted as *str*, called the *true string* of this index set, which is obtained by reading the bits in the content according to the indices. *str* can be hashed using a hash function denoted as *hash*, and the output is referred to as the hash of the index set. To construct a puzzle, the verifier needs  $L$  index sets denoted as  $I_1, \dots, I_L$ , where an index set is obtained by randomly choosing the indices, allowing repeat. The verifier randomly chooses one index set among the  $L$  index sets, denoted as  $I_{\hat{\ell}}$ , called the *answer index set*. It uses *hash* to get the hash of  $I_{\hat{\ell}}$ , denoted as  $\hat{h}$ , which is called the *hint* of the puzzle. The puzzle is basically the  $L$  index sets and  $\hat{h}$ . When challenged with a puzzle, the prover should prove that it knows which index set hashes into  $\hat{h}$ , by presenting another hash of  $I_{\hat{\ell}}$  generated by hash function *ans*. The purpose of using *ans* is to reduce the communication cost, as *str <sub>$\hat{\ell}$</sub>*  may be long. The verifier may issue  $z$  puzzles to the prover and the prover has to solve the all puzzles before a time threshold  $\theta$ .

From a high level, the strengths of the puzzle are (1) a prover has to know the content, otherwise it cannot get the true strings of the index sets (2) even if the prover knows the content, it still needs to spend time to try different index sets until it finds an index set with the same hash as the hint, refereed to as a confirm event, because the hash function is one-way. In practice, the verifier need not generate all index sets; it need only generate and find the hash of the answer index set. The verifier should not send the  $L$  index sets to the prover because this requires a large communication cost; instead, the verifier and the prover can agree on the same pseudorandom functions to generate the index sets and the verifier sends only a key for the pseudorandom functions. Therefore, this construction has low computation cost and low communication cost.

As an example, suppose  $n = 8$  and the content is 00110101. Suppose  $k = 4$ ,  $L = 3$ , and the three index sets in the puzzle are  $I_1 = \{5, 3, 7, 0\}$ ,  $I_2 = \{1, 2, 6, 3\}$ , and  $I_3 = \{2, 3, 5, 3\}$ . Correspondingly, *str<sub>1</sub>* = 1110, *str<sub>2</sub>* = 0101 and *str<sub>3</sub>* = 1111. Suppose the verifier chooses  $\hat{\ell} = 1$ . Suppose *hash* is the simply the the parity bit of the string, such that  $\hat{h} = 1$ . The prover

$q_{\text{hash}}$	The number of hash queries allowed, determined by $\theta$
$\Omega$	A special oracle for hash and content queries
$V$	The maximum number of missed bits
$\delta$	A positive number determined by puzzle parameters

TABLE 2  
LIST OF NOTATIONS IN THE PROOF

receives the the hint and generates the three index sets, and finds that only  $I_1$  has parity bit 1. Suppose *ans* is simply the parity bit of every pair of adjacent bits. The prover presents ‘01’ which proves that it knows  $I_1$  is the answer index set.

### B. Detailed Puzzle Construction

In the construction, it is assumed that the keys of the pseudorandom functions and the output of the hash functions are both  $\kappa$  bits. In practice,  $\kappa = 160$  suffices.

Pseudorandom functions are used to generate the index sets. A pseudorandom function family  $\{f_K\}$  is a family of functions parameterized by a secret key. Roughly speaking, once initialized by a key, a pseudorandom function generates outputs that are indistinguishable from true random outputs. Two pseudorandom function families are used:  $\{f_K^1 : \{1, \dots, L\} \rightarrow \{0, 1\}^\kappa\}$  and  $\{f_K^2 : \{1, \dots, k\} \rightarrow \{1, \dots, n\}\}$ .

Two hash functions are used in the construction, *hash* and *ans*. *hash* is used to get the hint. It actually hashes the concatenation of a  $\kappa$ -bit key, a number in the range of  $[1, L]$ , and a  $k$ -bit string into  $\kappa$ -bits:  $\{0, 1\}^\kappa \times \{1, \dots, L\} \times \{0, 1\}^k \rightarrow \{0, 1\}^\kappa$ . To prove the security of the puzzle, *hash* is modeled as a random oracle [6]. The other hash function is *ans*:  $\{0, 1\}^k \rightarrow \{0, 1\}^\kappa$ . For *ans*, only collision-resistance is assumed.

As mentioned earlier, a puzzle consists of the hint  $\hat{h}$  and  $L$  index-sets. The verifier first randomly picks a  $\kappa$ -bit string as key  $K_1$ . Then it randomly picks a number  $\hat{\ell}$  from  $[1, L]$  as the index of the answer index set. With  $K_1$  and  $\hat{\ell}$ , it generates  $K_2^{\hat{\ell}} \leftarrow f_{K_1}^1(\hat{\ell})$ .  $K_2^{\hat{\ell}}$  is used as the key for  $f_{K_2^{\hat{\ell}}}^2$  to generate the indices in the answer index set:  $I_{\hat{\ell}} = \{f_{K_2^{\hat{\ell}}}^2(1) \dots f_{K_2^{\hat{\ell}}}^2(k)\}$ . The verifier then finds *str <sub>$\hat{\ell}$</sub>* . It then uses the concatenation of  $K_1$ ,  $\hat{\ell}$ , and *str <sub>$\hat{\ell}$</sub>*  as the input to *hash* and uses the output as  $\hat{h}$ :  $\hat{h} \leftarrow \text{hash}(K_1, \hat{\ell}, \text{str}_{\hat{\ell}})$ . Including  $K_1$  and  $\hat{\ell}$  ensures that the results of one puzzle-solving process cannot be used in the solving process of another puzzle, regardless of the content,  $k$ , and  $L$ . The prover can generate index sets in the same way as the verifier generates the answer index set, and can compare the hash of the index sets with the hint until a confirm is found. When the prover finds a confirm upon string *str <sub>$\ell$</sub>* , it returns *ans(str <sub>$\ell$</sub> )*.

## III. THE SECURITY BOUND

In this section, we derive the new bound for the bandwidth puzzle. Although the puzzle is designed to defend against colluding adversaries, we begin with the simple case when there is only one adversary given only one puzzle, because the proof for this simple case can be extended to the case when multiple adversaries are given multiple puzzles.

### A. Single Adversary with a Single Puzzle

Consider a single adversary challenged with one puzzle. We begin with assumptions and definitions. Some key proof parameters and notations are listed in Table 2.

1) *Assumptions and Definitions:* In the proof, we model hash and ans as random oracles and refer to them as the *hash oracle* and the *answer oracle*, respectively. Obtaining a bit in the content is also modeled as making a query to the *content oracle* denoted as content. The adversary is given access to hash, ans, and content. To model the computational constraint of the prover in the limited time  $\theta$  allowed to solve the puzzle, we assume the number of queries to hash is no more than  $q_{\text{hash}}$ . To ensure that honest provers can solve the puzzle,  $q_{\text{hash}} \geq L$ . However, we do not assume any limitations on the number of queries to content and ans. We refer a query to content as a *content query* and a query to hash a *hash query*. We use  $\mathcal{A}$  to denote the algorithm adopted by the adversary.

In our proof, we define a special oracle,  $\Omega$ , as an oracle that answers two kinds of queries, both the content query and the hash query. Let  $\mathcal{B}$  be an algorithm for solving the puzzle, when given access to the special oracle  $\Omega$  and the answer oracle ans. If  $\mathcal{B}$  makes a content query,  $\Omega$  simply replies with the content bit. In addition, it keeps the history of the content queries made. We say a hash query to  $\Omega$  is *informed* if there are no more than  $V$  bits missing in the index set and *uninformed* otherwise, where  $V$  is a proof parameter much smaller than  $k$ . If  $\mathcal{B}$  makes an informed hash query for  $I_\ell$ ,  $\Omega$  replies with the hash of  $I_\ell$ ; otherwise, it returns  $\emptyset$ . In addition, if  $\mathcal{B}$  makes more than  $L$  hash queries for the puzzle,  $\Omega$  will not answer further hash queries.

2) *Problem Formalization:* The questions we seek to answer is: given  $q_{\text{hash}}$ , if the adversary has a certain advantage in solving the puzzle, how many content queries it must make to content *on average*? In the context of p2p content distribution, this is analogous to giving a lower bound on the average number of bits a peer must have downloaded if it can pass the puzzle challenge with a certain probability. Note that we emphasize on the average number of bits because a deterministic bound may be trivial: if the adversary happens to pick the answer index set in the first attempt of hash queries, only  $k$  content queries are needed. However, the adversary may be lucky once but unlikely to be always lucky. Therefore, if challenged with a large number of puzzles, the average number of queries it makes to content must be above a certain lower bound, which is the bound we seek to establish.

In an earlier work [14], an upper bound was given on the expected number puzzles that can be solved if the adversary is allowed  $q_{\text{hash}}$  hash queries and a certain number of content queries. In this work, we remove assumption on the maximum number of content queries. With less assumptions, our proof is less restrictive and applies to more general cases. The new problem is different from the problem studied in [14], and new techniques are needed to establish the bound. Note that although the adversaries is allowed to download as many bits as they wish, they prefer to employ an intelligent algorithm to minimize the number of downloaded bits because their intention is to use collusion to avoid spending bandwidth. The new bound guarantees that, if the adversaries wishes to have a certain advantage in solving the puzzles, there exists a lower bound on the average number of bits they have to download, regardless of the algorithm they adopt.

3) *Proof Sketch:* A sketch of our proof is as follows. As it is difficult to derive the optimal algorithm the adversary may adopt, our proof is “indirect.” That is, by using  $\Omega$ , we introduce a simplified environment which is easier to reason about. We show that an algorithm can be found in the simplified environment with performance close to that of the best algorithm the adversary may adopt in the real environment. This provides a link between the simplified environment and the real environment: knowing the bound for the former, the bound for the latter is a constant away. We establish the performance bound of the optimal algorithm in the simplified environment, by showing that to solve the puzzle with certain probability, an algorithm must make a certain number of informed hash queries to  $\Omega$  and the average number of unique indices in the informed queries, i.e., the number of content queries, is bounded.

4) *Proof Details:* Given any algorithm  $\mathcal{A}$  the adversaries may adopt, we construct an algorithm  $\mathcal{B}_\mathcal{A}$  that employs  $\mathcal{A}$  and implements oracle queries for  $\mathcal{A}$ .  $\mathcal{B}_\mathcal{A}$  terminates when  $\mathcal{A}$  terminates, and returns what  $\mathcal{A}$  returns. When  $\mathcal{A}$  makes a query,  $\mathcal{B}_\mathcal{A}$  replies as follows:

---

**Algorithm 1**  $\mathcal{B}_\mathcal{A}$  answers oracle queries for  $\mathcal{A}$

---

- 1: When  $\mathcal{A}$  makes a query to content,  $\mathcal{B}_\mathcal{A}$  makes the same content query to  $\Omega$  and **returns** the result to  $\mathcal{A}$ .
  - 2: When  $\mathcal{A}$  makes a query to ans,  $\mathcal{B}_\mathcal{A}$  makes the same query to ans and **returns** the result to  $\mathcal{A}$ .
  - 3: When  $\mathcal{A}$  makes a query to hash for  $I_\ell$ :
    - 1)  $\mathcal{B}_\mathcal{A}$  checks whether  $\mathcal{A}$  has made exactly the same query before. If yes, it **returns** the same answer as the last time.
    - 2)  $\mathcal{B}_\mathcal{A}$  checks whether there are no less than  $V$  bits in  $I_\ell$  that have not been queried. If yes, it **returns** a random string.
    - 3)  $\mathcal{B}_\mathcal{A}$  checks whether it has made a hash query for  $I_\ell$  before. If no,  $\mathcal{B}_\mathcal{A}$  makes a hash query to  $\Omega$ . If confirm is obtained upon this query,  $\mathcal{B}_\mathcal{A}$  knows that  $I_\ell$  is the answer index set, and sends content queries  $\Omega$  to get the remaining bits in  $I_\ell$ .
    - 4) If  $I_\ell$  is not the answer index set,  $\mathcal{B}_\mathcal{A}$  **returns** a random string.
    - 5) If the string  $\mathcal{A}$  submitted is the true string of  $I_\ell$ ,  $\mathcal{B}_\mathcal{A}$  **returns** the hash of  $I_\ell$ .
    - 6)  $\mathcal{B}_\mathcal{A}$  **returns** a random string.
- 

Let  $\omega()$  denote the average number of bits received by an algorithm, where the average is taken over the random choices of the algorithm and the randomness of the puzzle. We have

*Theorem 3.1:* Let  $C_\mathcal{A}$  be the event that  $\mathcal{A}$  returns the correct answer when  $\mathcal{A}$  is interacting directly with content, hash and ans. Let  $C_{\mathcal{B}_\mathcal{A}}$  be the event that  $\mathcal{B}_\mathcal{A}$  returns the correct answer, when  $\mathcal{B}_\mathcal{A}$  is interacting with  $\Omega$  and ans. Then,

$$\mathbb{P}[C_{\mathcal{B}_\mathcal{A}}] \geq \mathbb{P}[C_\mathcal{A}] - \frac{q_{\text{hash}}}{2^V},$$

and

$$\omega[\mathcal{B}_\mathcal{A}] \leq \omega[\mathcal{A}] + \frac{Lkq_{\text{hash}}}{2^V} + V.$$

*Proof:* In our construction,  $\mathcal{B}_A$  employs  $\mathcal{A}$ , and answers oracle queries for  $\mathcal{A}$ . Denote the random process of  $\mathcal{A}$  when it is interacting directly with content, hash and ans as  $W$ , and denote the random process of  $\mathcal{A}$  when it is interacting with the oracles implemented by  $\mathcal{B}_A$  as  $W'$ . We prove that  $W$  and  $W'$  will progress in the same way statistically with only one exception, while the probability of this exception is bounded.

First, we note that when  $\mathcal{A}$  makes a query to content or ans,  $\mathcal{B}_A$  simply gives the query result, therefore the only case needs to be considered is when  $\mathcal{A}$  makes a query to hash. When  $\mathcal{A}$  makes a query for  $I_\ell$  to hash,

- If there are still no less than  $V$  unknown bits in this index set,  $\mathcal{B}_A$  will simply return a random string, which follows the same distribution as the output of the hash modeled as a random oracle. If  $\ell \neq \hat{\ell}$ , such a query will not result in a confirm, and this will have same effect on the progress of the algorithm statistically as when  $\mathcal{A}$  is making a query to hash. However, if  $\ell = \hat{\ell}$ , it could happen that  $\mathcal{A}$  is making a query with the true string. In this case, the exception occurs. That is,  $W'$  will not terminate, but  $W$  will terminate with the correct answer to the puzzle. However, the probability of this exception is bounded from the above by  $\frac{q_{\text{hash}}}{2^V}$ , because if no less than  $V$  bits are unknown, the probability of making a hash query with the true string is no more than  $\frac{q_{\text{hash}}}{2^V}$ .
- If  $\mathcal{B}_A$  has made enough content queries for this index set,  $\mathcal{B}_A$  checks whether it has made hash query for this index set before. If no,  $\mathcal{B}_A$  makes the hash query, and if a confirm is obtained,  $\mathcal{B}_A$  knows that this is the answer index set and get the possible remaining bits in it; otherwise  $\mathcal{B}_A$  knows that it is not the answer index set. If  $I_\ell$  is not the answer index set,  $\mathcal{B}_A$  will simply return a random string, which will have the same effect statistically on the progress of  $\mathcal{A}$  as when  $\mathcal{A}$  is interacting with hash. If  $I_\ell$  is the answer index set,  $\mathcal{B}_A$  checks whether  $\mathcal{A}$  is submitting the true string, and returns the true hash if yes and a random string otherwise. This, clearly, also has the same effect statistically of the progress of  $\mathcal{A}$  as when  $\mathcal{A}$  is interacting with hash.

From the above discussion, we can see that  $P[C_{\mathcal{B}_A}]$  is no less than  $P[C_A]$  minus the probability of the exception. Therefore, the first half of the theorem is proved. We can also see that if the exception occurs,  $\mathcal{B}_A$  makes at most  $Lk$  more content queries than  $\mathcal{A}$ . If the exception does not occur,  $\mathcal{B}_A$  receives at most  $V$  bits than  $\mathcal{A}$  it encapsulates, and therefore at most  $V$  bits more than  $\mathcal{A}$  on average when  $\mathcal{A}$  is interacting directly with content, hash and ans. ■

Theorem 3.1 allows us to establish a connection between the “real” puzzle solver and the puzzle solver interacting with  $\Omega$ . The advantage of introducing  $\Omega$  is that a good algorithm will not send any uninformed queries to  $\Omega$ , because it will get no information from such queries. If there is a bound on the number of hash queries, which are all informed, it is possible to establish a lower bound on the number of unique indices involved in such queries, with which the lower bound of the puzzle can be established. It is difficult to establish such bound

based on hash directly because hash answers any queries. Although some queries are “more informed” than others, all queries have non-zero probabilities to get a confirm. The next theorem establishes the lower bound on the expected number of informed hash queries to achieve a given advantage by an optimal algorithm interacting with  $\Omega$ .

*Theorem 3.2:* Suppose  $\mathcal{B}$  is an optimal algorithm for solving the puzzle when interacting with  $\Omega$ . If  $\mathcal{B}$  solves the puzzle with probability no less than  $\epsilon$ , on average, the number of informed hash queries it makes is no less than  $\frac{(\epsilon - \frac{1}{2^V})(L+1)}{2}$ .

*Proof:* Let correct denote the event that  $\mathcal{B}$  returns the correct answer. Note that

$$\begin{aligned} P[\text{correct}] &= P[\text{correct} \mid \text{confirm}] P[\text{confirm}] \\ &\quad + P[\text{correct} \mid \neg \text{confirm}] P[\neg \text{confirm}] \\ &= P[\text{confirm}] \\ &\quad + P[\text{correct} \mid \neg \text{confirm}] P[\neg \text{confirm}] \\ &\leq P[\text{confirm}] + P[\text{correct} \mid \neg \text{confirm}] \\ &\leq P[\text{confirm}] + \frac{1}{2^V} \end{aligned}$$

Note that  $P[\text{correct} \mid \neg \text{confirm}] \leq \frac{1}{2^V}$  because if the algorithm returns the correct answer, it must have the true string of the answer index set, since ans is collision-resistant. If a confirm was not obtained, the answer index set is missing no less than  $V$  bits, since otherwise an optimal algorithm should make query which will result in a confirm. Therefore, the probability that the algorithm can obtain the true string of the answer index set is no more than  $\frac{1}{2^V}$ . Note that hash queries to  $\Omega$  will not help in the guessing of the true string, because  $\Omega$  is aware of the number of missing bits and will not reply with any information. Therefore, any algorithm that achieves advantage  $\epsilon$  in solving the puzzle must have an advantage of no less than  $\epsilon - \frac{1}{2^V}$  to get confirm.

Let  $P_1$  be the probability that  $\mathcal{B}$  makes no hash query and let  $P_i$  be the probability that  $\mathcal{B}$  stops making hash queries after all previous queries (queries 1 to  $i-1$ ) failed to generate a confirm for  $2 \leq i \leq L$ . Consider the probability that a confirm is obtained upon the  $i_{th}$  query. For a given set of  $P_1, P_2, \dots, P_L$ , because  $\hat{\ell}$  is picked at random, the probability is

$$\begin{aligned} &(1 - P_1) \frac{L-1}{L} (1 - P_2) \frac{L-2}{L-1} \dots (1 - P_i) \frac{1}{L-i} \\ &= \frac{1}{L} \prod_{j=1}^i (1 - P_j) \end{aligned}$$

Therefore, the probability that the algorithm can get a confirm is

$$\sum_{i=1}^L \left[ \frac{1}{L} \prod_{j=1}^i (1 - P_j) \right].$$

The event that exactly  $i$  queries are made occurs when a confirm was obtained upon the  $i_{th}$  query, or when all first  $i$  queries failed to obtain the confirm and the algorithm decides to stop making queries. The probability is thus

$$\left[ \prod_{j=1}^i (1 - P_j) \right] \left[ \frac{1}{L} + \frac{L-i}{L} P_{i+1} \right].$$

Note that  $P_{L+1}$  is not previously defined. However, as  $\frac{L-i}{L} = 0$  when  $i = L$ , for convenience, we can use the same expression for all  $1 \leq i \leq L$  for any arbitrary value of  $P_{L+1}$ . To derive the lower bound, we therefore need to solve the problem of minimizing

$$\sum_{i=1}^L i \left[ \prod_{j=1}^i (1 - P_j) \right] \left[ \frac{1}{L} + \frac{L-i}{L} P_{i+1} \right]$$

subject to constraint that

$$\sum_{i=1}^L \left[ \frac{1}{L} \prod_{j=1}^i (1 - P_j) \right] = \epsilon - \frac{1}{2V}$$

and

$$0 \leq P_i \leq 1.$$

To solve the problem, we let  $\eta_i = \prod_{j=1}^i (1 - P_j)$  and note that  $P_{i+1} = 1 - \frac{\eta_{i+1}}{\eta_i}$ . Therefore,

$$\begin{aligned} & \sum_{i=1}^L i \left[ \prod_{j=1}^i (1 - P_j) \right] \left[ \frac{1}{L} + \frac{L-i}{L} P_{i+1} \right] \\ &= \sum_{i=1}^L i \eta_i \left[ \frac{1}{L} + \frac{L-i}{L} \left( 1 - \frac{\eta_{i+1}}{\eta_i} \right) \right] \\ &= \frac{1}{L} \left[ \sum_{i=1}^L (L-i+1) \eta_i - \sum_{i=1}^{L-1} (L-i) \eta_{i+1} \right] \\ &= \frac{1}{L} \left[ \sum_{i=1}^L (L-i+1) \eta_i \right]. \end{aligned}$$

We therefore consider a new problem as minimizing  $\frac{1}{L} \left[ \sum_{i=1}^L (L-i+1) \eta_i \right]$  subject to constraint that  $\sum_{i=1}^L \eta_i = L(\epsilon - \frac{1}{2V})$ ,  $0 \leq \eta_i \leq 1$ ,  $\eta_{i+1} \leq \eta_i$ . The optimal value for the newly defined problem must be no more than that of the original problem, because any valid assignment of  $\{P_i\}_i$  gives a valid assignment of  $\{\eta_i\}_i$ . To achieve the optimal value of the new problem, note that if  $i < j$ , the coefficient of  $\eta_i$  is more than  $\eta_j$  in the objective function, therefore, to minimize the objective function, we should reduce  $\eta_i$  and increase  $\eta_j$ . Considering that  $\{\eta_i\}_i$  is nondecreasing, the optimal is achieved when all  $\eta_i$  are set to the same value  $(\epsilon - \frac{1}{2V})$ , and the optimal value is  $\frac{(\epsilon - \frac{1}{2V})(L+1)}{2}$ . ■

Based on Theorem 3.2, any algorithm with an advantage of  $\epsilon$  must make no less than certain number of informed hash queries to  $\Omega$  on average. We next derive the number of unique indices in a given number index sets. We first need the following lemma.

**Lemma 3.3:** Suppose  $c$  indices are randomly picked among  $n$  indices, with repeat. Let  $Y$  be the random number denoting the number of unique indices among  $c$  indices. Let  $\mu = n(1 - \delta)[1 - (1 - \frac{1}{n})^c]$  for a constant  $0 < \delta < 1$  and  $\eta = \frac{c - n \ln \frac{n}{n-\mu}}{n \sqrt{(\frac{1}{n-\mu} - \frac{1}{n})}}$ .

We have

$$\mathbb{P}[Y \leq \mu] \leq \frac{e^{-\eta^2/2}}{\sqrt{2\pi\eta}}$$

when  $n \rightarrow \infty$  and  $c \rightarrow \infty$ .

*Proof:* Consider the process when indices are randomly taken from  $n$  indices. Let  $Z_i$  be random number denoting the

number of samples needed to get the  $i_{th}$  unique index. Clearly,  $\mathbb{P}[Z_1 = 1] = 1$ . In general, note that  $Z_i$  follows the geometric distribution, i.e.,

$$\mathbb{P}[Z_i = j] = \left(\frac{i-1}{n}\right)^{j-1} \frac{n-i+1}{n}.$$

Let  $p_i = \frac{n-i+1}{n}$ , we have  $\mathbb{E}[Z_i] = \frac{1}{p_i}$ , and  $\text{Var}[Z_i] = \frac{1-p_i}{p_i^2}$ . Also,  $\{Z_i\}_i$  are independent of each other. Define  $S_\mu = \sum_{i=1}^\mu Z_i$  and note that  $\mathbb{P}[Y \leq \mu] = \mathbb{P}[S_\mu \geq c]$ . Therefore, we will focus on finding  $\mathbb{P}[S_\mu \geq c]$ .

Define  $Z'_i = Z_i - \mathbb{E}[Z_i]$ . Let  $S'_\mu = \sum_{i=1}^\mu Z'_i$  and note that  $\mathbb{P}[S_\mu \geq c] = \mathbb{P}\left[S'_\mu \geq c - \sum_{i=1}^\mu \frac{1}{p_i}\right]$ . As  $\{Z'_i\}_i$  are independent random variables with zero mean, due to the Central Limit Theorem,  $S'_\mu$  approximately follows the Gaussian distribution with zero mean and variance  $\sum_{i=1}^\mu \frac{1-p_i}{p_i^2}$ . Note that as  $n \rightarrow \infty$  and  $c \rightarrow \infty$ ,  $\mu \rightarrow \infty$ , therefore,

$$\mathbb{P}\left[S'_\mu \geq c - \sum_{i=1}^\mu \frac{1}{p_i}\right] = Q\left(\frac{c - \sum_{i=1}^\mu \frac{1}{p_i}}{\sqrt{\sum_{i=1}^\mu \frac{1-p_i}{p_i^2}}}\right),$$

where  $Q()$  is the Gaussian Error Integral. To simplify the result, note that

$$\sum_{i=1}^\mu \frac{1}{p_i} = \frac{n}{n} + \frac{n}{n-1} + \dots + \frac{n}{n-\mu+1} \leq n \ln \frac{n}{n-\mu},$$

and

$$\begin{aligned} \sum_{i=1}^\mu \frac{1-p_i}{p_i^2} &= \sum_{i'=0}^{\mu-1} \frac{1 - \frac{n-i'}{n}}{(\frac{n-i'}{n})^2} = \sum_{i'=0}^{\mu-1} \frac{ni'}{(n-i')^2} \\ &= n \sum_{i'=0}^{\mu-1} \left[ \frac{n}{(n-i')^2} - \frac{n-i'}{(n-i')^2} \right] \\ &< n^2 \sum_{i'=0}^{\mu-1} \frac{1}{(n-i')^2} \\ &< n^2 \left( \frac{1}{n-\mu} - \frac{1}{n} \right) \end{aligned}$$

Applying these bounds, we have

$$\mathbb{P}[Y \leq \mu] \leq Q\left(\frac{c - n \ln \frac{n}{n-\mu}}{n \sqrt{(\frac{1}{n-\mu} - \frac{1}{n})}}\right)$$

The proof completes since  $Q(x) \leq \frac{e^{-x^2/2}}{\sqrt{2\pi x}}$  [17]. ■

Note that according to the well-known coupon collector problem,  $n[1 - (1 - \frac{1}{n})^c]$  is actually the average number of unique indices among  $c$  indices, and  $\delta$  determines how far  $\mu$  deviates from this value. This lemma establishes the bound of the probability that the number of unique indices is less than  $1 - \delta$  fraction of the average.

Let  $Y_s^J$  denote the random variable of the minimum number of unique indices in  $s$  index sets among all possible choices of  $s$  index sets picked from  $J$  index sets. Let  $\mu_s = n(1 - \delta)[1 - (1 - \frac{1}{n})^{sk}]$  and  $\eta_s = \frac{sk - n \ln \frac{n}{n-\mu_s}}{n \sqrt{(\frac{1}{n-\mu_s} - \frac{1}{n})}}$ , due to Lemma 3.3, we have

$$\mathbb{P}[Y_s^J \leq \mu_s] \leq \frac{e^{-\eta_s^2/2}}{\sqrt{2\pi\eta_s}} J^s$$

as  $n \rightarrow \infty$  and  $sk \rightarrow \infty$ . This is because event  $Y_s^J \leq \mu_s$  happens if one combination of  $s$  index sets have no more than  $\mu_s$  unique indices, which happens with probability as given in Lemma 3.3, and the total number of combinations to pick  $s$  index sets is less than  $J^s$ .

Considering practical puzzles, we note that  $n$  is very large, e.g.,  $10^7$ , as well as  $k$ , e.g.,  $10^4$ . For any  $s \geq 1$ , as  $n \rightarrow \infty$  and  $sk \rightarrow \infty$ ,  $(1 - \frac{1}{n})^{sk}$  approaches  $e^{-sk/n}$ . We also pick puzzle parameters as well as  $\delta$ , such that  $\mathbb{P}[Y_s^J \leq \mu_s]$  is negligibly small for conceivable values of  $J$ ; see Section IV for details which has been confirmed by numerical analysis. Therefore, we guarantee that for the puzzles we use, *the number of unique indices in any  $s$  index sets is no less than  $(1-\delta)n(1-e^{-sk/n})$  with overwhelming probability.*

The next lemma is needed in determining the average number of unique indices in a certain average number of index sets.

**Lemma 3.4:** Consider a linear programming problem of maximizing  $\sum_{i=0}^L P_i e^{-id}$  subject to the constraint that  $\sum_{i=0}^L P_i i = \beta$ ,  $\sum_{i=0}^L P_i = \gamma$ ,  $P_i \geq 0$ , where  $0 \leq d$ ,  $0 \leq \gamma \leq 1$  and  $0 \leq \beta \leq \gamma L$ . Denote the optimal value of the objective function as  $F_L(\beta, \gamma)$ , we have

$$F_L(\beta, \gamma) = \gamma - \frac{(1 - e^{-Ld})}{L} \beta$$

That is, to achieve the optimal value is to let  $P_0 = \gamma - \frac{\beta}{L}$  and  $P_L = \frac{\beta}{L}$ , while let  $P_i = 0$  for  $0 < i < L$ .

*Proof:* We will induction on  $L$ . To begin with, consider when  $L = 1$ . In this case, due to the constraints,  $P_0$  and  $P_1$  can be uniquely determined as  $P_0 = \gamma - \beta$  and  $P_1 = \beta$ . Therefore,

$$F_1(\beta, \gamma) = \gamma - \beta + \beta e^{-d},$$

and the lemma is true. Suppose the lemma is true till  $j$ . To get  $F_{j+1}(\beta, \gamma)$ , suppose  $P_{j+1} = \gamma - \gamma'$ , where  $\gamma' \leq \gamma$ . Given this,  $\beta' = \sum_{i=0}^j P_i i = \beta - (\gamma - \gamma')(j+1)$ , and therefore,

$$\begin{aligned} F_{j+1}(\beta, \gamma) &= F_j(\beta', \gamma') + (\gamma - \gamma')e^{-(j+1)d} \\ &= \gamma' - \frac{1 - e^{-jd}}{j} \beta' + (\gamma - \gamma')e^{-(j+1)d} \\ &= \gamma' \left[ 1 - \frac{(1 - e^{-jd})(j+1)}{j} - e^{-(j+1)d} \right] - \\ &\quad \frac{1 - e^{-jd}}{j} [\beta - \gamma(j+1)] + \gamma e^{-(j+1)d} \end{aligned}$$

Regarding  $\gamma'$  as a variable, its coefficient is

$$1 - \frac{(1 - e^{-jd})(j+1)}{j} - e^{-(j+1)d},$$

which is no more than 0. To see this, consider function

$$f(x) = 1 - \frac{(1 - e^{-jx})(j+1)}{j} - e^{-(j+1)x}.$$

Note that  $f(0) = 0$ , and  $f'(x) < 0$  when  $x \geq 0$ . Therefore, to maximize the objective function,  $\gamma'$  should be as small as possible. Note that  $jP_j = \beta - (j+1)\gamma + (j+1)\gamma'$  and  $P_0 + P_j = \gamma'$ . Therefore,

$$P_0 = \frac{(j+1)\gamma - \beta - \gamma'}{j}.$$

Since  $P_0 \leq \gamma'$ , we have  $\gamma' \geq \gamma - \frac{\beta}{j+1}$ . Therefore,  $F_{j+1}(\beta, \gamma) = \gamma - \frac{\beta}{j+1} + \frac{\beta}{j+1} e^{-(j+1)d}$ . ■

Suppose we randomly pick index sets from a total  $L$  index sets when the average number of picked index sets is  $\beta$ . Let  $P_i$  denote the probability that  $i$  index sets are picked, where  $0 \leq i \leq L$ . We now give the lower bound of the average number of unique indices in the index sets picked, denoted as  $U(\beta)$ . We have

$$U(\beta) \geq \sum_{i=0}^L (1 - \delta)n(1 - e^{-ik/n})P_i.$$

Therefore, to derive the lower bound is to maximize  $\sum_{i=0}^L P_i e^{-ik/n}$  subject to the constraints  $\sum_{i=0}^L P_i i = \beta$ ,  $\sum_{i=0}^L P_i = 1$ ,  $P_i \geq 0$ .

Based on Lemma 3.4, we immediately have

**Lemma 3.5:** If the average number of picked index sets is  $\beta$ , then

$$U(\beta) \geq \frac{(1 - \delta)n(1 - e^{-Lk/n})\beta}{L}$$

where  $\delta$  is a parameter determined by the puzzle parameters.

We may finally assemble the parts together. Suppose  $\mathcal{A}$  has an advantage of  $\sigma$  in solving the puzzle when receiving  $\omega(\mathcal{A})$  bits on average. Based on Theorem 3.1,  $\mathcal{B}_\mathcal{A}$  has an advantage of no less than  $\sigma - \frac{q_{\text{hash}}}{2^V}$  while receiving no more than  $\omega(\mathcal{A}) + \frac{Lkq_{\text{hash}}}{2^V} + V$  bits on average. Based on Theorem 3.2, to achieve an advantage of at least  $\sigma - \frac{q_{\text{hash}}}{2^V}$ , an algorithm must make at least  $\frac{(\sigma - \frac{q_{\text{hash}} + 1}{2^V})(L+1)}{2}$  hash queries. Based on Lemma 3.5, also considering that  $\mathcal{B}$  needs to receive only  $k - V + 1$  bits per index set,  $\mathcal{B}$  receives at least  $U(\frac{(\sigma - \frac{q_{\text{hash}} + 1}{2^V})(L+1)}{2}) - L(V - 1)$  bits on average. Therefore,

**Theorem 3.6:** Suppose  $\mathcal{A}$  solves the puzzle with probability no less than  $\sigma$ . Let  $\omega(\mathcal{A})$  denote the average number of received bits. We have

$$\begin{aligned} \omega(\mathcal{A}) &\geq \frac{(1 - \delta)n(1 - e^{-Lk/n})(\sigma - \frac{q_{\text{hash}} + 1}{2^V})(L+1)}{2L} \\ &\quad - L(V - 1) - \frac{Lkq_{\text{hash}}}{2^V} - V \end{aligned}$$

where  $q_{\text{hash}}$ ,  $V$ , and  $\delta$  are constants determined by the puzzle parameters.

## B. Multiple Adversaries with Multiple Puzzles

We next consider the more complicated case when multiple adversaries are required to solve multiple puzzles. Suppose there are  $A$  adversaries, and the number of puzzles they attempt to solve is  $P$ . Note that  $P$  is greater than  $A$  when  $z > 1$ .

1) *Proof Sketch:* The proof uses the same idea as the single adversary case. Basically, we extend  $\Omega$  to handle multiple adversaries, where  $\Omega$  gives correct answer to a hash query from an adversary only if the number of bits the adversary received for the index set is greater than  $k - V$ , *regardless of the number of bits other adversaries received*. With similar arguments as the single adversary case, we can establish the relationship between the algorithm performance when interacting with  $\Omega$  and with the real oracles. We also obtain the average number of informed queries the adversaries must make to achieve certain

advantages when interacting with  $\Omega$ . The bound is established after solving several optimization problems.

2) *Proof Details:* Suppose the adversaries run an algorithm  $\mathcal{A}$  that solves the  $P$  puzzles with probability  $\sigma$  while receiving  $\omega(\mathcal{A})$  bits on average. We wish to bound from below  $\omega(\mathcal{A})$  for a given  $\sigma$ . We extend the definition of  $\Omega$  and let it remember the content queries from each adversary. We use  $I_\ell^p$  to denote index set  $\ell$  in puzzle  $p$  where  $1 \leq p \leq P$ . If an adversary makes a hash query for  $I_\ell^p$  while this adversary has made content query for more than  $k - V$  bits in  $I_\ell^p$ ,  $\Omega$  replies with the hash of  $I_\ell^p$ , otherwise, it returns  $\emptyset$ . In addition, if  $\mathcal{B}$  makes more than  $L$  hash queries for a particular puzzle,  $\Omega$  will not answer further hash queries for this puzzle.

Similar to the single puzzle case, given an algorithm  $\mathcal{A}$  for solving the puzzles, we construct an algorithm  $\mathcal{B}_\mathcal{A}$  employing  $\mathcal{A}$  denoted as  $\mathcal{B}_\mathcal{A}$ .  $\mathcal{B}_\mathcal{A}$  terminates when  $\mathcal{A}$  terminates, and returns what  $\mathcal{A}$  returns. Algorithm 2 describes how  $\mathcal{B}_\mathcal{A}$  implements oracle queries for  $\mathcal{A}$  which is very similar to the single adversary case.

---

**Algorithm 2**  $\mathcal{B}_\mathcal{A}$  answers oracle queries for  $\mathcal{A}$

---

- 1: When  $\mathcal{A}$  makes a query to content,  $\mathcal{B}_\mathcal{A}$  makes the same content query to  $\Omega$  and gives the result to  $\mathcal{A}$ .
  - 2: When  $\mathcal{A}$  makes a query to ans,  $\mathcal{B}_\mathcal{A}$  makes the same query to ans and gives the result to  $\mathcal{A}$ .
  - 3: When  $\mathcal{A}$  makes a query for  $I_\ell^p$  to hash at adversary  $v$ :
    - 1)  $\mathcal{B}_\mathcal{A}$  checks whether adversary  $v$  has made exactly the same query before. If yes, it **returns** the same answer.
    - 2)  $\mathcal{B}_\mathcal{A}$  checks whether there are no less than  $V$  bits in  $I_\ell^p$  that have not been queried for at adversary  $v$ , and if this is true, it **returns** a random string.
    - 3)  $\mathcal{B}_\mathcal{A}$  checks if it has made a hash query for  $I_\ell^p$ , if no, it makes a hash query to  $\Omega$ . If confirm is obtained upon this query,  $\mathcal{B}_\mathcal{A}$  knows  $I_\ell^p$  is the answer index set of puzzle  $p$ .  $\mathcal{B}_\mathcal{A}$  sends content queries  $\Omega$  to get the remaining bits in  $I_\ell^p$ .
    - 4) If  $I_\ell^p$  is not the answer index set of puzzle  $p$ ,  $\mathcal{B}_\mathcal{A}$  **returns** a random string.
    - 5) If the string  $\mathcal{A}$  submitted is the true string of  $I_\ell^p$ ,  $\mathcal{B}_\mathcal{A}$  **returns** the hash of  $I_\ell^p$ .
    - 6)  $\mathcal{B}_\mathcal{A}$  **returns** a random string.
- 

With very similar arguments as in Theorem 3.1, we can have

*Theorem 3.7:* Let  $C_\mathcal{A}$  be the event that  $\mathcal{A}$  returns the correct answers when it is interacting directly with content, hash and ans. Let  $C_{\mathcal{B}_\mathcal{A}}$  be the event that  $\mathcal{B}_\mathcal{A}$  returns the correct answers, when it is interacting with  $\Omega$  and ans. Then,

$$\mathbb{P}[C_{\mathcal{B}_\mathcal{A}}] \geq \mathbb{P}[C_\mathcal{A}] - \frac{Aq_{\text{hash}}}{2^V},$$

and

$$\omega[\mathcal{B}_\mathcal{A}] \leq \omega[\mathcal{A}] + \frac{PLkAq_{\text{hash}}}{2^V} + VP.$$

Let  $\mathcal{B}$  denote the optimal algorithm for solving the puzzles when the algorithm is interacting with  $\Omega$ . Note that if the

probability that  $\mathcal{B}$  solves all puzzles is no less than  $\epsilon$ , the probability that an individual puzzle is solved is no less than  $\epsilon$ . Based on Theorem 3.2, if a puzzle is solved with probability no less than  $\epsilon$ , the average number of hash queries made for this puzzle is no less than  $\frac{(\epsilon - \frac{1}{2^V})(L+1)}{2}$ . There are  $P$  puzzles, and we obtain the following theorem due to the linearity of expectation.

*Theorem 3.8:* If the probability that  $\mathcal{B}$  solves all puzzles is no less than  $\epsilon$ , on average, the number of informed hash queries is no less than

$$\frac{P(\epsilon - \frac{1}{2^V})(L+1)}{2}$$

Next, we wish to bound from below the number of unique indices if the adversaries collectively have to make  $T$  informed hash queries. Here we define the unique indices at adversary  $v$  as the total number of unique indices in the index sets that it made hash queries for, and denote it as  $u_v$ . The total number of unique indices is defined as  $\sum_{v=1}^A u_v$ . Recall that  $\Omega$  will not answer a hash query from adversary  $v$  if adversary  $v$  has not received enough number bits for this index set. Note that  $\Omega$  will not answer the hash query even if there exists another adversary knowing enough bits for this index set. In other words, content queries made at one adversary do not count as content queries at other adversaries, which is the one of the key differences between the single adversary case and the multiple adversary case.  $\mathcal{B}$  may be able to assign hash queries to the adversaries intelligently, such that  $\sum_{v=1}^A u_v$  is minimized. For instance, if two index sets share a large number of indices, they should be assigned to the same adversary. Nevertheless, we have

*Lemma 3.9:* If the number of informed queries made by  $\mathcal{B}$  is  $T$ ,

$$\sum_{v=1}^A u_v \geq (1 - \delta)n[t(1 - e^{-q_{\text{hash}}k/n}) + (1 - e^{-(T - q_{\text{hash}}t)k/n})],$$

where  $t = \lfloor T/q_{\text{hash}} \rfloor$  and  $\lfloor x \rfloor$  denotes the largest integer no more than  $x$ .

*Proof:* An adversary may make no more than  $q_{\text{hash}}$  queries. Suppose the number of hash queries made by adversary  $v$  is  $s_v$ . We have

$$\sum_{v=1}^A u_v \geq \sum_{v=1}^A (1 - \delta)n(1 - e^{-s_v k/n})$$

Therefore, to minimize  $\sum_{v=1}^A u_v$  is to maximize  $\sum_{v=1}^A e^{-s_v k/n}$  subject to the constraints that  $\sum_{v=1}^A s_v = T$  and  $0 \leq s_v \leq q_{\text{hash}}$ .

We claim that the optimal is achieved when  $s_i$  is set to be  $q_{\text{hash}}$  for  $1 \leq i \leq \lfloor T/q_{\text{hash}} \rfloor$ , which we show by induction on the number of adversaries. First consider when  $A = 2$ . If  $T \leq q_{\text{hash}}$ , we claim that  $\sum_{i=1}^2 e^{-s_i k/n}$  is maximized when  $s_1 = T$  and  $s_2 = 0$ , which is because for any valid  $s_1$  and  $s_2$ ,

$$\begin{aligned} & (1 + e^{-Tk/n}) - (e^{-s_1 k/n} + e^{-s_2 k/n}) \\ &= (1 - e^{-s_1 k/n})(1 - e^{-s_2 k/n}) \\ &\geq 0. \end{aligned}$$

Similarly, if  $q_{\text{hash}} \leq T \leq 2q_{\text{hash}}$ ,  $\sum_{i=1}^A e^{-s_i k/n}$  is maximized when  $s_1 = q_{\text{hash}}$  and  $s_2 = T - q_{\text{hash}}$ , which is because for any valid  $s_1$  and  $s_2$ ,

$$\begin{aligned} & (e^{-q_{\text{hash}} k/n} + e^{-(T-q_{\text{hash}})k/n}) - (e^{-s_1 k/n} + e^{-s_2 k/n}) \\ &= (1 - e^{(-s_1 + T - q_{\text{hash}})k/n}) (e^{-T + q_{\text{hash}} k/n} - e^{-s_2 k/n}) \\ &\geq 0. \end{aligned}$$

Therefore our claim is true for  $A = 2$ . Suppose our claim is true for  $A = j$ . For  $A = j + 1$ , suppose in the optimal assignment,  $s_{j+1} = 0$ . Then, our claim is true based on the induction hypothesis. If in the optimal assignment,  $s_{j+1} > 0$ , we prove that in the optimal assignment  $s_i = q_{\text{hash}}$  for all  $1 \leq i \leq j$ , therefore our claim is still true. This because if  $s_i < q_{\text{hash}}$  for some  $i$ , we can increase  $s_i$  while decreasing  $s_{j+1}$ . Using similar arguments as for case when  $A = 2$ , this will increase the objective function, thus violating the fact that the assignment is optimal. ■

Similar to the single adversary case, if the average number of informed hash queries  $\mathcal{B}$  makes is  $\beta$ , we need to bound from below the average number of unique indices in the involved index sets, denoted as  $U(\beta)$ .

*Lemma 3.10:* Consider there are  $A$  adversaries given  $P$  puzzles, if the average number of informed hash queries is  $\beta$ ,

$$U(\beta) \geq \frac{(1 - \delta)n\beta(1 - e^{-q_{\text{hash}} k/n})}{q_{\text{hash}}}$$

*Proof:* Denote the probability that there are  $i$  queries as  $P_i$ , where  $0 \leq i \leq PL$ . For notational simplicity, in this proof, we let  $d = k/n$ . Based on Lemma 3.9, we want to bound from below

$$\sum_{i=0}^{PL} P_i [(t_i - 1)(1 - e^{-q_{\text{hash}} d}) + (1 - e^{-[i - q_{\text{hash}}(t_i - 1)]d})]$$

under the constraints that

$$\begin{aligned} \sum_{i=0}^{PL} P_i i &= \beta, \\ \sum_{i=0}^{PL} P_i &= 1, \\ P_i &\geq 0, \end{aligned}$$

where  $t_i$  is an integer such that  $(t_i - 1)q_{\text{hash}} \leq i < t_i q_{\text{hash}}$ . To solve this problem, suppose  $C$  is the minimum integer satisfying  $PL \leq Cq_{\text{hash}} - 1$ , we relax the problem to minimizing

$$\Psi = \sum_{i=0}^{Cq_{\text{hash}}-1} P_i [(t_i - 1)(1 - e^{-q_{\text{hash}} d}) + (1 - e^{-[i - q_{\text{hash}}(t_i - 1)]d})]$$

under the same constraints that

$$\begin{aligned} \sum_{i=0}^{Cq_{\text{hash}}-1} P_i i &= \beta, \\ \sum_{i=0}^{Cq_{\text{hash}}-1} P_i &= 1, \\ P_i &\geq 0. \end{aligned}$$

The optimal of relaxed problem will be no more than the optimal of the original problem. To solve the relaxed problem, let

$$\Psi_s = \sum_{i=(s-1)q_{\text{hash}}}^{sq_{\text{hash}}-1} P_i [(s-1)(1 - e^{-q_{\text{hash}} d}) + (1 - e^{-[i - q_{\text{hash}}(s-1)]d})]$$

for  $1 \leq s \leq C$ . Clearly,  $\Psi = \sum_{s=1}^C \Psi_s$ .

Suppose a set of  $\{\gamma_s\}_s$  and  $\{\beta_s\}_s$  are given where  $\sum_{s=1}^C \gamma_s = 1$  and  $\sum_{s=1}^C \beta_s = \beta$ . The set of  $\{\gamma_s\}_s$  and  $\{\beta_s\}_s$  are called *feasible* if it is possible to find  $\{P_i\}_i$  such that

$$\sum_{i=(s-1)q_{\text{hash}}}^{sq_{\text{hash}}-1} P_i = \gamma_s,$$

and

$$\sum_{i=(s-1)q_{\text{hash}}}^{sq_{\text{hash}}-1} P_i i = \beta_s$$

for all  $1 \leq s \leq C$ . Note that  $\{\gamma_s\}_s$  and  $\{\beta_s\}_s$  are feasible if and only if

$$(s-1)q_{\text{hash}}\gamma_s \leq \beta_s \leq (sq_{\text{hash}} - 1)\gamma_s.$$

When  $\{\gamma_s\}_s$  and  $\{\beta_s\}_s$  are given and are feasible, to minimize  $\Psi$  is to minimize each individual  $\Psi_s$ . Note that

$$\begin{aligned} \Psi_s &= \gamma_s [(s-1)(1 - e^{-q_{\text{hash}} d}) + 1] \\ &\quad - \sum_{i=(s-1)q_{\text{hash}}}^{sq_{\text{hash}}-1} P_i e^{-[i - q_{\text{hash}}(s-1)]d} \\ &= \gamma_s [(s-1)(1 - e^{-q_{\text{hash}} d}) + 1] \\ &\quad - \sum_{h=0}^{q_{\text{hash}}-1} P_{q_{\text{hash}}(s-1)+h} e^{-hd}, \end{aligned}$$

where  $h = i - q_{\text{hash}}(s-1)$ . Note that if

$$\sum_{i=(s-1)q_{\text{hash}}}^{sq_{\text{hash}}-1} P_i i = \beta_s,$$

then

$$\sum_{h=0}^{q_{\text{hash}}-1} P_{q_{\text{hash}}(s-1)+h} h = \beta_s - (s-1)q_{\text{hash}}\gamma_s.$$

Denote the minimum value of  $\Psi_s$  for given  $\gamma_s$  and  $\beta_s$  as  $\Psi_s^{\gamma_s, \beta_s}$ . Applying Lemma 3.4,

$$\begin{aligned} \Psi_s^{\gamma_s, \beta_s} &= \gamma_s [(s-1)(1 - e^{-q_{\text{hash}} d}) + 1] - \gamma_s \\ &\quad + \left[ \frac{1 - e^{-(q_{\text{hash}}-1)d}}{q_{\text{hash}} - 1} \right] [\beta_s - (s-1)q_{\text{hash}}\gamma_s] \\ &= \gamma_s (s-1) \left[ \frac{q_{\text{hash}} e^{-(q_{\text{hash}}-1)d}}{q_{\text{hash}} - 1} - e^{-q_{\text{hash}} d} \right. \\ &\quad \left. - \frac{1}{q_{\text{hash}} - 1} \right] + \left[ \frac{1 - e^{-(q_{\text{hash}}-1)d}}{q_{\text{hash}} - 1} \right] \beta_s \end{aligned}$$

Let

$$a = \frac{q_{\text{hash}} e^{-(q_{\text{hash}}-1)d}}{q_{\text{hash}} - 1} - e^{-q_{\text{hash}} d} - \frac{1}{q_{\text{hash}} - 1}$$



and

$$b = \frac{1 - e^{-(q_{\text{hash}}-1)d}}{q_{\text{hash}} - 1},$$

we have

$$\Psi \geq \sum_{s=1}^C \Psi_s^{\gamma_s, \beta_s} = b\beta + a \sum_{s=1}^C (s-1)\gamma_s.$$

We also note that  $a < 0$ , which is because function

$$f(x) = \frac{q_{\text{hash}} e^{-(q_{\text{hash}}-1)x}}{q_{\text{hash}} - 1} - e^{-q_{\text{hash}}x} - \frac{1}{q_{\text{hash}} - 1}$$

is 0 when  $x = 0$ , while  $f'(x) < 0$  for  $x > 0$ . Therefore, finding the minimum value of  $\Psi$  is equivalent to finding a set of feasible  $\{\gamma_s\}_s$  and  $\{\beta_s\}_s$  such that  $\sum_{s=1}^C (s-1)\gamma_s$  is maximized.

We consider the problem of maximizing

$$W = \sum_{s=1}^C (s-1)\gamma_s$$

subject to the constraints that

$$\begin{aligned} (s-1)q_{\text{hash}}\gamma_s &\leq \beta_s \leq (sq_{\text{hash}} - 1)\gamma_s, \\ \sum_{s=1}^C \gamma_s &= \gamma, \\ \sum_{s=1}^C \beta_s &= \beta, \end{aligned}$$

where  $0 \leq \gamma \leq 1$  and  $0 \leq \beta \leq \gamma(Cq_{\text{hash}} - 1)$ . Denote the maximum value of  $W$  as  $W^*$ . We claim that

- If  $(C-1)q_{\text{hash}}\gamma < \beta$ ,  $W^* = \gamma(C-1)$  and the optimal is achieved when  $\gamma_C = \gamma$ ,  $\beta_C = \beta$ , while  $\gamma_s = 0$  and  $\beta_s = 0$  for  $1 \leq s < C$ ;
- If  $(C-1)q_{\text{hash}}\gamma \geq \beta$ ,  $W^* = \frac{\beta}{q_{\text{hash}}}$ , and the optimal value is achieved when  $\gamma_1 = \gamma - \frac{\beta}{(C-1)q_{\text{hash}}}$ ,  $\beta_1 = 0$ ,  $\gamma_C = \frac{\beta}{(C-1)q_{\text{hash}}}$ ,  $\beta_C = \beta$ , while  $\gamma_s = 0$  and  $\beta_s = 0$  for  $1 < s < C$ .

To show this, we use induction on  $C$ . First, when  $C = 2$ ,  $W = \gamma_2$ . Note that

- If  $q_{\text{hash}}\gamma < \beta$ , we can let  $\gamma_2 = \gamma$ ,  $\beta_2 = \beta$ , while  $\gamma_1 = 0$  and  $\beta_1 = 0$ , in which case  $\gamma_2$  is maximized, while all constraints are satisfied;
- If  $q_{\text{hash}}\gamma \geq \beta$ , note that for any given  $\beta_2$ ,  $\gamma_2 \leq \frac{\beta_2}{q_{\text{hash}}} \leq \frac{\beta}{q_{\text{hash}}}$ . When  $\beta \leq q_{\text{hash}}\gamma$ , we may let  $\gamma_1 = \gamma - \frac{\beta}{q_{\text{hash}}}$ ,  $\beta_1 = 0$ ,  $\gamma_2 = \frac{\beta}{q_{\text{hash}}}$ ,  $\beta_2 = \beta$ , such that all constraints are satisfied, while  $\gamma_2$  is maximized.

Therefore, our claim is true when  $C = 2$ . Suppose the claim is true till  $C = j$ . When  $C = j+1$ ,

- If  $j q_{\text{hash}}\gamma < \beta$ , we may let  $\gamma_{j+1} = \gamma$ ,  $\beta_{j+1} = \beta$ , and let  $\gamma_s = 0$  and  $\beta_s = 0$  for  $1 \leq s \leq j$ , such that all constraints are satisfied. In this case,  $W = j\gamma$ . Since  $W \leq j\gamma$ , we have  $W^* = j\gamma$ .
- If  $j q_{\text{hash}}\gamma \geq \beta$ , suppose some  $0 \leq \gamma' \leq \gamma$ ,  $0 \leq \beta' \leq \beta$  are given that also satisfy

$$(\gamma - \gamma')j q_{\text{hash}} \leq (\beta - \beta') \leq (\gamma - \gamma')[(j+1)q_{\text{hash}} - 1]$$

and

$$\beta' \leq \gamma'(j q_{\text{hash}} - 1).$$

We can let  $\sum_{s=1}^j \gamma_s = \gamma'$  and  $\sum_{s=1}^j \beta_s = \beta'$ .

- If  $(j-1)q_{\text{hash}}\gamma' < \beta'$ , based on the induction hypothesis, the maximum value of  $\sum_{s=1}^j (s-1)\gamma_s$  is  $(j-1)\gamma'$ , and hence

$$W \leq j\gamma - \gamma'.$$

Because

$$(\gamma - \gamma')j q_{\text{hash}} \leq (\beta - \beta'),$$

we have

$$\gamma' \geq \gamma - \frac{\beta}{j q_{\text{hash}}} + \frac{\beta'}{j q_{\text{hash}}}.$$

As  $(j-1)q_{\text{hash}}\gamma' < \beta'$ , we have

$$\gamma j - \frac{\beta}{q_{\text{hash}}} < \gamma'.$$

Therefore,

$$W \leq \frac{\beta}{q_{\text{hash}}}.$$

- If  $(j-1)q_{\text{hash}}\gamma' \geq \beta'$ , based on the induction hypothesis, the maximum value of  $\sum_{s=1}^j (s-1)\gamma_s$  is  $\frac{\beta'}{q_{\text{hash}}}$ , and hence

$$W \leq \frac{\beta'}{q_{\text{hash}}} + (\gamma - \gamma')j.$$

Since  $(\gamma - \gamma')j \leq \frac{\beta - \beta'}{q_{\text{hash}}}$ , we have

$$W \leq \frac{\beta}{q_{\text{hash}}}.$$

Note that  $W$  achieves  $\frac{\beta}{q_{\text{hash}}}$  when  $\gamma_1 = \gamma - \frac{\beta}{j q_{\text{hash}}}$ ,  $\beta_1 = 0$ ,  $\gamma_{j+1} = \frac{\beta}{j q_{\text{hash}}}$ ,  $\beta_{j+1} = \beta$ , while  $\gamma_s = 0$  and  $\beta_s = 0$  for  $1 < s \leq j$ . Therefore,  $W^* = \frac{\beta}{q_{\text{hash}}}$ .

Note that actually, in the first case when  $j q_{\text{hash}}\gamma < \beta$ ,  $j\gamma \leq \frac{\beta}{q_{\text{hash}}}$ , therefore we also have  $W^* \leq \frac{\beta}{q_{\text{hash}}}$ . Hence,

$$\Psi \geq b\beta + a \frac{\beta}{q_{\text{hash}}} = \frac{\beta(1 - e^{-q_{\text{hash}}d})}{q_{\text{hash}}},$$

which completes our proof.  $\blacksquare$

Similar to single puzzle case, we may now put things together. Suppose  $\mathcal{A}$  has an advantage of  $\sigma$  in solving the puzzles when receiving  $\omega(\mathcal{A})$  bits on average. Based on Theorem 3.7,  $\mathcal{B}_{\mathcal{A}}$  has an advantage of no less than  $\sigma - \frac{Aq_{\text{hash}}}{2V}$  while receiving no more than  $\omega(\mathcal{A}) + \frac{PLkAq_{\text{hash}}}{2V} + VP$  bits on average. Based on Theorem 3.8, to achieve an advantage of at least  $\sigma - \frac{Aq_{\text{hash}}}{2V}$ , any algorithm must make at least  $\frac{P(\sigma - \frac{Aq_{\text{hash}}}{2V})(L+1)}{2}$  hash queries. Based on Lemma 3.10, also considering that  $\mathcal{B}$  needs to receive only  $k - V + 1$  bits per index set,  $\mathcal{B}$  receives at least  $U(\frac{P(\sigma - \frac{Aq_{\text{hash}}}{2V})(L+1)}{2}) - PL(V-1)$  bits on average. Therefore,

*Theorem 3.11:* Suppose  $\mathcal{A}$  adversaries are challenged with  $P$  puzzles. Suppose  $\mathcal{A}$  solves the puzzle with probability no

less than  $\sigma$  and let  $\omega(\mathcal{A})$  denote the average number of received bits. We have

$$\omega(\mathcal{A}) \geq \frac{(1 - \delta)nP(\sigma - \frac{q_{\text{hash}} + 1}{2^V})(L + 1)(1 - e^{-q_{\text{hash}}k/n})}{2q_{\text{hash}} - PL(V - 1) - \frac{PLkAq_{\text{hash}}}{2^V} - VP}$$

where  $q_{\text{hash}}$ ,  $V$ , and  $\delta$  are constants determined by the puzzle parameters.

#### IV. DISCUSSIONS

In this section we discuss the bound and its practical implications. We begin by considering a simple strategy the adversaries may adopt to be compared with the bound.

##### A. A Simple Adversary Strategy

We note that there exists a simple strategy the adversaries may adopt to be compared with the bound. In this strategy, when challenged with the puzzles, the adversaries flip a coin and decide whether to attempt to solve the puzzles. They attempt with probability  $\sigma$ ; otherwise they simply ignore the puzzles. If they decide to solve the puzzles, the adversaries select  $\frac{P(L+1)}{2q_{\text{hash}}}$  members, and let each of them get the entire content. Each of the chosen adversaries makes  $q_{\text{hash}}$  hash queries allowed for them. For each puzzle, the adversaries make hash queries for the index sets one by one until a confirm is obtained.

We now analyze the performance of this strategy. We argue that the adversaries can solve the puzzles with probability close to 1 if they decide to attempt, hence their advantage is  $\sigma$ . Note that to get a confirm for puzzle according to this strategy, the number of hash queries follows a uniform distribution in  $[1, L]$  and is independent of other puzzles. The total number of hash queries is a random variable with mean  $\frac{P(L+1)}{2}$ . As the number of puzzles increases, the distribution of this variable approaches a Gaussian distribution centered around the mean with decreasing variance. Therefore, if the adversaries can make  $\frac{P(L+1)}{2}$  hash queries, the probability that they can solve the puzzles asymptotically approaches 1. Note that this is possible because there are  $\frac{P(L+1)}{2q_{\text{hash}}}$  selected adversaries, each making  $q_{\text{hash}}$  queries. According to this strategy, the average number of bits downloaded is  $\frac{\sigma n P(L+1)}{2q_{\text{hash}}}$ .

##### B. Puzzle Parameter Space

As can be observed in Theorem 3.11, the dominating factor in the number of bits is roughly  $\frac{\sigma n P(L+1)}{2q_{\text{hash}}}$ , if the following conditions are satisfied:

- 1)  $\delta$  is small comparing to 1,
- 2)  $e^{-q_{\text{hash}}k/n}$  is small comparing to 1,
- 3)  $2^V$  is much larger than  $Aq_{\text{hash}}$ ,
- 4)  $2^V$  is no less than  $kAq_{\text{hash}}$ ,
- 5)  $V$  is much smaller than  $k$ .

If these conditions are satisfied, the bound approaches the actual number of bits downloaded by the simple adversary strategy above, therefore is tight.

We show that there are a wide range of values of  $L$ ,  $k$ ,  $z$ ,  $n$  and  $q_{\text{hash}}$  satisfying these conditions, for which the bound can be applied to provide security guarantees. Note that  $P = Az$ .

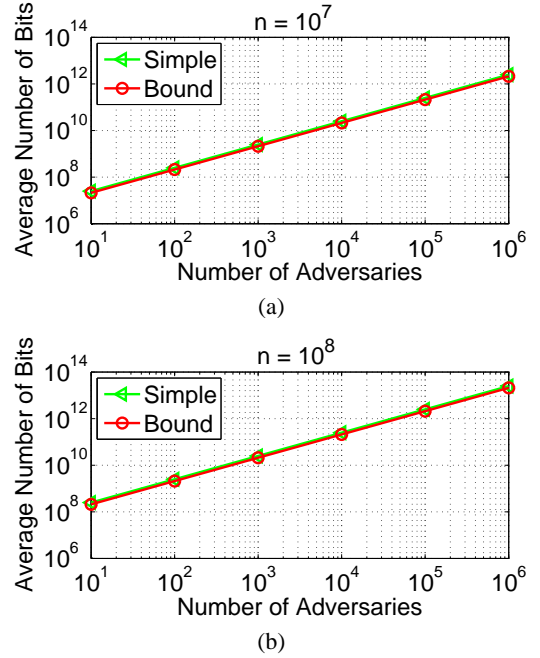


Fig. 1. Comparison of the average number of bits downloaded by the simple strategy and the bound when  $\sigma = 1$ ,  $k = 10^4$ ,  $q_{\text{hash}} = 4n/k$  and  $Lz = q_{\text{hash}}/2$ . (a).  $n = 10^7$ . (b).  $n = 10^8$ .

In the following we use an example to illustrate the choice of parameters when  $A \leq 10^6$ ; the parameters can be similarly determined for other values of  $A$ . Concerning the conditions,

- For Condition 1, we note that  $\delta$  should be as small as possible, provided that the probability that the number of unique indices in any  $s$  index sets is less than  $(1 - \delta)n(1 - e^{-sk/n})$  is negligibly small. We find numerically that when  $A \leq 10^6$ , for  $k \geq 10^4$ ,  $zL \leq 10^6$ ,  $n \geq 10^7$ , if  $\delta = 0.1$ , this probability for any  $s$  is below  $10^{-12}$ .
- Condition 2 can be considered as satisfied when  $q_{\text{hash}}k \geq 4n$ , noting that  $e^{-4} = 0.018$ .
- When  $A \leq 10^6$ ,  $V$  can be set to be 60. Condition 3 is satisfied when  $q_{\text{hash}} \leq 10^6$ . Condition 4 is satisfied if  $k \leq 10^6$ . Condition 5 is satisfied if  $k \geq 10^4$ .

The above discussions give the range of the puzzle parameters. Basically, if we let  $\delta = 0.1$  and  $V = 60$ , we only require  $n \geq 10^7$ ,  $10^6 \geq k \geq 10^4$ ,  $10^6 \geq Lz$ ,  $q_{\text{hash}}k \geq 4n$ ,  $10^6 \geq q_{\text{hash}}$ , when  $A \leq 10^6$ .

Note that  $k$  should be set to its lower bound  $10^4$ , because a larger value of  $k$  results in a heavier load of the verifier.  $q_{\text{hash}}$  should be no less than  $Lz$  to ensure an honest prover can solve the puzzles. Figure 1 shows the average number of bits needed by the simple strategy and the lower bound as a function of the number of adversaries for different content sizes, when  $\sigma = 1$ ,  $q_{\text{hash}} = 4n/k$  and  $Lz = q_{\text{hash}}/2$ . We can see that they differ only by a small constant factor. We have tested other parameters satisfying the constraints and the results show similar trends.

##### C. Puzzle Parameters in Practice

We also note that the parameter space is not restrictive in practice. Considering the speed of modern communication networks, a reasonable rate to challenge the prover should be

machine	CPU	SHA1	AES
pc3000	3.0GHz 64-bit	202165	4059157
pc2000	2.0GHz	71016	2605490
pc850	850MHz	39151	1086667
pc600	600MHz	29064	789624

TABLE 3  
SHA-1 AND AES FUNCTION CALLS EXECUTED IN ONE SECOND.

once at least every 1MB of data, i.e., when  $n$  is at least around  $10^7$ . There is also no obstacle to set  $k$  to be  $10^4$ . Concerning  $\theta$  which determines  $q_{\text{hash}}$ , note that the puzzles should be solved in a reasonable amount of time to reduce the load of the prover, but the time should also be non-trivial to account for random fluctuations of network latency. Therefore a reasonable value of  $\theta$  should be in the order of several seconds. Given these choices of  $n$ ,  $k$ , and  $\theta$ ,  $z$  and  $L$  should satisfy three conditions, according to our earlier discussions: (1)  $kL \leq 10^6$ , (2)  $kLz$  should be no less than, for example,  $2n$ , and (3) the time to make  $kL$  hash queries should be several seconds but no more than  $\theta$ .

Note that there are two time consuming tasks when making hash queries, which are the hash function call and the generation of the random indices. The choices of hash function and random number generator have been discussed in [14]. Basically, secure hash functions such as SHA-1 can be used as the hash function and block ciphers such as AES can be used to generate the random indices. The optimization of the puzzle implementation is out of the scope of this paper due to the limit of space. We here show the speed of several machines in Emulab [1] when executing the SHA-1 hash and the AES encryption in the Openssl library [2], summarized in Table 3, when the input to SHA-1 is  $10^4$  bits and the AES is 128 bits. If  $n = 10^7$ ,  $k = 10^4$  and  $\theta = 3\text{sec}$ , the results indicate that on modern mainstream machines such as pc3000 and pc2000, (1) it is not possible to make more than  $10^6$  hash queries within  $\theta$ , (2) it is possible to make enough number of hash queries within  $\theta$  such that  $kLz \geq 2n$  after optimizations in random index generation, and (3) when  $kLz \geq 2n$ , solving the puzzles will take time in the order of seconds.

## V. RELATED WORK

Using puzzles has been proposed (e.g., in [10], [12], [3], [9], [8]) to defend against email spamming or denial of service attacks. In these schemes, the clients are required to spend time to solve puzzles before getting access to the service. The purpose of the bandwidth puzzle is to verify whether the claimed content transactions took place, where the ability to solve the puzzles is tied to the amount contents actually downloaded. As such, the construction of the bandwidth puzzle is different from existing puzzles.

Proofs of data possession (PDP) (e.g., [4], [11], [5]) and Proofs of retrievability (POR) (e.g., [13], [7], [15]) have been proposed to allow a client to verify whether the data has been modified in a remote store. As discussed in [14], the key differences between PDP/POR schemes and the bandwidth puzzle include the following. First, PDP/POR assumes a single verifier and prover, while the bandwidth puzzle considers one verifier with many potentially colluding provers. Second, the

bandwidth puzzle has low computational cost at the verifier, which is desirable in the case when one verifier has to handle many provers, while the existing PDP/POR schemes may incur heavy computational cost at the verifier. The proof techniques for PDP/POR schemes are also different from the techniques used in this paper, because collusion is not considered in existing PDP/POR schemes.

The bandwidth puzzle was first proposed in [14] along with a detailed performance evaluation based on an implemented p2p streaming system and a larger simulated p2p network. The results show that the bandwidth puzzle can effectively improve the performance of the honest users in the presence of colluding adversaries. A bound was also given in [14] on the expected number of puzzles solved when the adversaries can make no more than a certain number of hash queries and download no more than a certain number of content bits. The purpose of this work is to derive a bound on the average number of bits downloaded, when the adversaries can make no more than a certain number of hash queries but can download as many content bits as they wish. Therefore the problem studied in this work is different from that in [14]. Our analysis show that the new bound is asymptotically tight for all numbers of adversaries, while the bound given in [14] deteriorates quickly as the number of adversaries increases, and the largest number of adversaries used in [14] is 50 when evaluating the bound. As discussed earlier, it is not difficult to find puzzle parameters satisfying the requirements of the new bound. The bound given in [14] requires much more restrictive choices of parameters. For instance, the suggested values of  $k$  and  $L$  are  $\frac{1}{4}n^{3/10}$  and  $\frac{1}{12}n^{71/100}$ , respectively (it was assumed that  $z = 1$  in [14]). A consequence is that the limit on number of bits downloaded by the adversaries must be significantly smaller than  $n$  for the bound to give satisfactory results.

## VI. CONCLUSIONS

In this paper, we proved a new bound on the performance of the bandwidth puzzle which has been proposed to defend against colluding adversaries in p2p content distribution networks. Our proof is based on reduction, and gives the lower bound of the average number of downloaded bits to achieve a certain advantage by the adversaries. The bound is asymptotically tight in the sense that it is a small fraction away from the average number of bits downloaded when following a simple strategy. The new bound is a significant improvement over the existing bound which was derived under more restrictive conditions and much looser. The improved bound can be used to guide the choice of puzzle parameters to improve the performance of practical systems.

## REFERENCES

- [1] Emulab. <http://www.emulab.net>.
- [2] Openssl. <http://www.openssl.org/>.
- [3] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology*, 5:299–327, 2005.
- [4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proc. of ACM CCS*, 2007.
- [5] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and Efficient Provable Data Possession. *IACR eArchive* 2008/114 at <http://eprint.iacr.org/2008/114.pdf>, 2008.

- [6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, Nov. 1993.
- [7] K. Bowers, A. Juels, and A. Oprea. Proofs of Retrievability: Theory and Implementation. IACR eArchive 2008/175 at <http://eprint.iacr.org/2008/175.pdf>, 2008.
- [8] S. Doshi, F. Monrose, and A. Rubin. Efficient memory bound puzzles using pattern databases. In *Proceedings of the International Conference on Applied Cryptography and Network Security*, 2006.
- [9] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Proceedings of CRYPTO 2003*, Aug. 2003.
- [10] C. Dwork and M. Naor. Pricing via processing, or, combatting junk mail. In *Advances in Cryptology – CRYPTO '92 (Lecture Notes in Computer Science 740)*, pages 139–147, 1993.
- [11] D. L. G. Filho and P. S. L. M. Barreto. Demonstrating data possession and uncheatable data transfer. IACR eArchive 2006/150 at <http://eprint.iacr.org/2006/150.pdf>, 2006.
- [12] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of the 6th ISOC Network and Distributed System Security Symposium*, Feb. 1999.
- [13] A. Juels and B. S. K. Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Oct. 2007.
- [14] M. Reiter, V. Sekar, C. Spensky, and Z. Zhang. Making contribution-aware p2p systems robust to collusion attacks using bandwidth puzzles. In *Proc. of ICISS*, [http://www.cs.cmu.edu/~vyass/papers/paper\\_44.pdf](http://www.cs.cmu.edu/~vyass/papers/paper_44.pdf), 2009.
- [15] H. Shacham and B. Waters. Compact Proofs of Retrievability. IACR eArchive 2008/073 at <http://eprint.iacr.org/2008/073.pdf>, 2008.
- [16] Y. Sung, M. Bishop, and S. Rao. Enabling Contribution Awareness in an Overlay Broadcasting System. In *Proc. ACM SIGCOMM*, 2006.
- [17] J. M. Wozencraft and I. M. Jacobs. *Principles of Communication Engineering*. Wiley, 1965.